Niveau(x) d'enseignement: 0 Numéro EDUQ : 8570294
Centre d'intérêt: 640

Bélisle, Laurent.

Les langages de programmation : le langage LOGO / animateur, Laurent Bélisle, co-animateur, Jean-Pierre Thibault ; [élaboré par Laurent Bélisle à partir d'un document écrit par Jean-François Cloutier et Alain Taurisson et d'un autre document écrit par Michel Arcourt ; [pour le] Service de la formation du personnel scolaire, Direction générale du développement pédagogique. -- [Québec] : La Direction, 1985.
ii, 78 p. : graph. ; 28 cm. -- (Guide de formation / Programme de formation lourde en micro-informatique appliquée à l'éducation ; 6.2)

Titre de la couv.

Descripteur(s): *Informatique *Programmation *Application pédagogique de l'ordinateur *Guide de l'enseignant

Identificateur(s) de sujet: *Langage de programmation *LOGO (Langage de programmation)

Le guide de formation, produit dans le cadre du "Programme de formation lourde en micro-informatique appliqué à l'éducation", fait partie d'une série comprenant une vingtaine de documents. Ces guides ont été produits pour les agents mutiplicateurs comme documents d'accompagnement aux différents cours donnés dans le cadre de la formation lourde. L'objectif du document est de sensibiliser et familiariser les parti_ipants au langage de programmation LOGO. Il comprend sept parties: 1. LOGO: un langage de programmation ou une philosophie de l'éducation; 2. LOGO et le développement intellectuel de l'élève; 3. LOGO et l'informatique; 4. généralités sur le LOGO; 5. activités de création; 6. exploration d'un micro-monde; 7. exercice supplémentaire de résolution de problèmes.

Type de document: d

EDUQ 8570294

SERVICE DE LA FORMATION DU PERSONNEL SCOLAIRE

PROGRAMME DE FORMATION LOURDE
EN MICRO-INFORMATIQUE APPLIQUEE A L'EDUCATION

GUIDE DE FORMATION

VOLET

LES LANGAGES DE PROGRAMMATION

SECTION

LE LANGAGE LOGO

ANIMATEUR/TRICE

LAURENT BELISLE

CO-ANIMATEUR/TRICE

JEAN-PIERRE THIBAULT

JANVIER 1985

6.2

Gouvernement du Québec
Ministère de l'Éducation
Direction générale
du développement pédagogique

EDUQ 8570294

Ce document a été élaboré par

LAURENT BELISLE

à partir d'un document écrit par

JEAN-FRANCOIS CLOUTIER et ALAIN TAURISSON

et d'un autre document écrit par

MICHEL ARCOUET

TABLE DES MATIERES

Page

1. LOGO: UN LANGAGE DE PROGRAMMATION OU UNE PHILOSOPHIE DE L'EDUCATION 1

1.1 Un peu d'histoire 1
1.2 Les fondements de LOGO 1

2. LOGO ET LE DEVELOPPEMENT INTELLECTUEL DE L'ELEVE 3

2.1 L'espace et la tortue 3
2.2 LOGO et la construction de modèles 4
2.3 LOGO et l'heuristique 6
2.4 LOGO et la construction d'algorithmes 7

3. LOGO ET L'INFORMATIQUE 9

3.1 LOGO est un langage structuré 9
3.2 LOGO et la programmation structurée 10

4. GENERALITES SUR LE LOGO 13

4.1 Le chargement du langage en mémoire 13
4.2 Le vocabulaire 13
4.3 La syntaxe 14
4.4 L'éditeur LOGO 16
4.5 Les écrans du LOGO 17

5. ACTIVITES DE CREATION 19

5.1 Du triangle à la spirale 19
5.2 La récursivité 25
5.3 Le traitement de listes 28

Page

6. EXPLORATION D'UN MICRO-MONDE 33

6.1 Les micro-mondes et l'apprentissage 33

6.2 Une description du micro-monde de la tortue tri-dimensionnelle 34

6.3 Une aide à l'exploration: les fiches de travail ... 37

6.4 Le résumé des commandes du micro-monde de la tortue tri-dimensionnelle 48

7. EXERCICE SUPPLEMENTAIRE DE RESOLUTION DE PROBLEMES 51

7.1 L'acquisition d'habiletés de résolution de problèmes 51

7.2 Le problème à résoudre: modéliser le comportement olfactif d'une mouche 52

7.3 Les réponses au problème de la modélisation du comportement de la mouche 72

1. LOGO: UN LANGAGE DE PROGRAMMATION OU UNE PHILOSOPHIE DE L'EDUCATION

1.1 Un peu d'histoire

Vers la fin des années 1950, début des années 1960, furent inventés des langages de programmation évolués, appelés langages de troisième génération. Ceux-ci se distinguent des précédents en au moins deux aspects: la langue des codes et la gestion de la mémoire.

En effet, d'une part, ces nouveaux langages utilisent maintenant des mots tirés de la langue de l'usager au lieu de simples codes binaires ou hexadécimaux. Il devient donc nécessaire d'implanter dans l'ordinateur un programme qui traduit les programmes écrits avec un tel langage quasi humain en un code utilisable par la machine. Ce sont les interpréteurs ou les compilateurs.

D'autre part, on confie en même temps à l'interpréteur ou au compilateur le soin de gérer l'utilisation de la mémoire: la place réservée aux variables, les répétitions...

Au moment de cette révolution, les concepteurs eurent à faire des choix. Certains optèrent pour l'économie pour la machine et la simplicité: c'est le cas du langage BASIC. D'autres choisirent la performance, notamment pour traiter les problèmes de gestion: c'est le cas du langage COBOL. En APL, on choisit la performance pour traiter les problèmes dans lesquels les tableaux ont une grande importance. Les concepteurs du LISP voulurent plutôt créer un système proche d'une certaine conception de l'intelligence humaine. LOGO est directement issu de LISP.

1.2 Les fondements de LOGO

LOGO, inventé par Minsky et Papert au M.I.T., tire ces racines d'une conception piagétienne ou néo-piagétienne de la connaissance. LOGO est un langage de procédures où le programmeur est amené à nommer chacune des

actions qu'il veut faire exécuter par l'ordinateur, un peu comme on nomme les concepts que nous développons dans notre intelligence.

De même LOGO permet de construire de nouvelles actions plus complexes en utilisant les actions plus simples. Il suffit de les invoquer. Ceci se produit d'une façon analogue au développement des concepts: un concept existant peut être utilisé dans le développement d'un autre.

Par exemple, si on sait ce qu'est communiquer et comment communiquer, si on connaît également le fonctionnement des ordinateurs, alors il est probablement possible d'apprendre à communiquer par ordinateur. Notons, cependant, que les deux premiers concepts sont essentiels mais non suffisants pour permettre le développement du troisième. De même, une procédure LOGO peut utiliser d'autres procédures. De plus, chacune d'elle a une existence propre et peut être utilisée par d'autres. Cela ressemble à notre façon de créer des liens entre les concepts.

LOGO essaie d'imiter certaines autres façons de penser. Les actions séquentielles sont naturelles et ordonnées, sans besoin de numérotation. Les répétitions sont autorisées tant par la commande REPETE que par des boucles. Cependant, LOGO permet et valorise la récursivité qui est isomorphe à la si puissante pensée récurrente. Les diverses méthodes de répétition peuvent être vues comme des cas particuliers de la récursion. Enfin, les listes de propriétés en LOGO permettent de facilement faire des inférences logiques.

2. LOGO ET LE DEVELOPPEMENT INTELLECTUEL DE L'ELEVE

2.1 L'espace et la tortue

Les exercices de la présente session de deux jours, vont vous permettre de vous familiariser avec la tortue LOGO. Vous allez appréhender l'impact intéressant que cela peut avoir sur le développement des concepts spatiaux reliés au mouvement de la tortue. Vous allez voir que l'enfant, en programmant la tortue peut apprendre à construire des algorithmes.

Plus encore, l'enfant apprend à agir directement sur son modèle dans l'espace. En effet, ce n'est plus lui qui se déplace: il est immobile et c'est l'objet qu'il commande qui se déplace. Il doit prévoir à l'avance l'ensemble, ou, tout au moins, une partie du trajet. C'est beaucoup plus complexe que de se déplacer, évaluer où on est rendu et continuer.

Les déplacements de la tortue obéissent aux commandes AVANCE/RECULE et GAUCHE/DROITE. Le modèle de l'espace est "polaire". En effet, la tortue est au départ au centre de l'écran. Pour accéder à un point quelconque, elle doit pivoter d'un certain angle, puis, avancer d'une certaine distance. Cela revient à décrire un point du plan par sa distance au centre et par l'angle qu'il fait avec une droite de référence. C'est le système de coordonnées polaires.

Tout comme les humains, la tortue transporte avec elle son système de référence. On a donc un système de référence polaire relatif, en mouvement. On a aussi une géométrie "intrinsèque" et "locale".

Une propriété (géométrique) intrinsèque est une propriété qui ne dépend que de la figure elle-même et non de sa relation avec l'espace de référence. Une propriété (géométrique) locale est une propriété qui ne dépend que des points situés dans l'environnement immédiat plutôt que d'un point de référence éloigné. A titre d'illustration, mentionnons simplement la perception que les humains ont du système planétaire.

Tout ceci a pour conséquence que l'on peut utiliser des procédures pour décrire l'environnement plutôt que des équations. On a donc accès à des expressions dynamiques plutôt que statiques. Ainsi presque tous les enfants en viennent assez rapidement à ainsi décrire un cercle: REPETE 360 [AVANCE 1 DROITE 1]. Cette définition implique un passage à la limite. Le cercle est le cas limite du polygone régulier. Dans les cours de géométrie traditionnelle (synthétique, euclidienne), même avec des adolescents, ce passage ne se fait pourtant que très péniblement. De plus, les jeunes en viennent souvent à se rendre compte que la définition du cercle dépend de la qualité de l'affichage. Ceux-ci ont donc perçus que la limite dépend du seuil de tolérance. Autrement dit, si on utilise les expressions scientifiques du calcul différentiel et intégral, le epsilon dépend du delta.

LOGO met donc en place des connaissances et des expériences directement réinvestissables dans des processus mathématiques complexes, tels qu'on en retrouve dans le calcul avancé et dans la cinématique. Ceci, alors qu'au CEGEP on déplore justement l'absence de telles capacités chez un grand nombre d'élèves de niveau collégial.

2.2 *LOGO et la construction de modèles*

Malgré le fait que le stade formel, selon Piaget, n'est atteint qu'assez tard chez les enfants, LOGO favorise, de par sa structure, la construction de modèles, même chez ceux qui ne semblent pas avoir atteint ce niveau.

Considérons un exemple: le dessin de figures régulières par la tortue. Quelle est la séquence d'instructions qui permet de tracer un carré? Tout le monde trouve: AVANCE 100 DROITE 90 AVANCE 100 DROITE 90 AVANCE 100 DROITE 90 AVANCE 100. Et à la question "La tortue est-elle revenue en position initiale?", tout le monde répond "Non, il faut encore tourner de 90 degrés". Rendu à cette étape, les gens remarquent tous une régularité. On répète quatre fois le même mouvement: AVANCE 100 DROITE 90.

On peut donc utiliser: REPETE 4 [AVANCE 100 DROITE 90]. Maintenant, il reste à faire remarquer à l'enfant qu'un tour complet est de 360 degrés, que la tortue a donc effectué un tour complet.

Considérons un deuxième problème qui consiste à faire tracer par la tortue un triangle régulier (équilatéral). Mis à part les problèmes de calcul, les enfants trouvent assez rapidement: REPETE 3 [AVANCE 100 DROITE 120]. Notez que pour les adultes, ce problème est souvent plus difficile que pour les enfants, car ils essaient de le résoudre à partir de la géométrie euclidienne. Ils oublient que l'angle de rotation est l'angle extérieur.

Par la suite, tous les enfants sont en mesure de trouver comment on trace un pentagone régulier. Ceux qui ont appris à diviser, formulent le problème d'une façon équivalente à REPETE 5 [AVANCE 100 DROITE 360/5]. Ils peuvent même dire dans leurs propres mots qu'un polygone régulier à N côtés se trace ainsi: REPETE :N [AVANCE 100 DROITE 360/:N].

Les enfants partent donc de cas très particuliers et parviennent à se donner un modèle permettant de décrire tous les polygones réguliers. Ils construisent un modèle, une définition utilisant une variable. Cela est possible même avec des enfants du primaire.

De la même façon, les enfants apprennent rapidement à faire des procédures. Dès la deuxième année, certains enfants peuvent faire des procédures. Ils demandent comment on pourrait faire des carrés de "grandeur variable" (sic) avec la même procédure. Pourtant, le concept de variable n'est abordé en mathématique qu'à la fin du premier cycle du secondaire.

Une expérience de ce type, où on vient à une formulation unique pour décrire tous les polygones réguliers, ainsi que le cercle, se déroule assez rapidement avec des élèves du secondaire, même avec ceux qui sont réputés être lents. Elle met souvent plus du double du temps à se produire chez les adultes (enseignants). Un certain déconditionnement est probablement

nécessaire. Une telle généralisation est évidemment plus longue à se produire chez les élèves du primaire. Mais elle se produit dès la deuxième année. Ce phénomène se produit-il aussi facilement dans d'autres contextes?

Cet exemple montre bien que les enfants peuvent avancer très tôt d'un point de vue du développement formel. Il suffit qu'ils soient placés dans des situations favorables. LOGO est actuellement un des plus efficaces micro-mondes le permettant.

2.3 *LOGO et l'heuristique*

LOGO se veut le reflet de la manière de penser des personnes. C'est aussi, par conséquent, un langage proche de la façon de résoudre des problèmes. Dans les activités de programmation en LOGO, les élèves exercent un grand nombre de stratégies et d'heuristiques. Ces stratégies et heuristiques sont répertoriées dans le deuxième objectif général du programme d'études I.S.I. Des exemples apportés précédemment, on peut dégager les stratégies suivantes que les élèves utilisent:

- procéder par essais et erreurs,
- dégager des informations relatives à un problème,
- trouver des cas particuliers,
- généraliser,
- définir,
- construire un modèle,
- généraliser un modèle,
- formuler,
- communiquer.

En LOGO, il est facile d'observer l'exercice de ces stratégies et heuristiques. C'est parce qu'elles s'exercent facilement.

2.4 LOGO et la construction d'algorithmes

LOGO est un langage modulaire et extensible. En effet, on peut naturellement et facilement regrouper sous un nom un ensemble d'actions. Pour faire exécuter cet ensemble, il suffit d'invoquer le nom. On a donc des procédures. Celles-ci sont autonomes et, du point de vue de l'usager, elles ont le même statut que les instructions primitives. C'est pourquoi on dit que le langage est extensible. Il peut être augmenté.

Cette propriété favorise grandement la construction d'algorithmes, car chaque partie de programme donne lieu à une procédure qui peut être construite et testée séparément. Il devient facile d'identifier et isoler les diverses structures informatiques impliquées: répétitions, choix, variables...

On a observé au XEROX PARC (Palo Alto Research Center) que très peu de personnes, même chez les adultes, peuvent construire des programmes de plus de deux pages. Cependant, un grand nombre de personnes sont capables de construire des ensembles de petits modules dont la fusion atteindra plusieurs pages.

Cette structure modulaire favorise aussi l'analyse descendante. Celle-ci consiste à prendre une idée générale et à la spécifier en sous-idées. Chacune d'elles est alors traitée comme une idée principale qu'il faut de nouveau spécifier. On descend ainsi, par paliers, vers une formulation très détaillée. Cette méthode est l'une des plus efficaces dans la construction d'algorithmes. En utilisant cette façon de penser, les enfants deviennent très efficaces.

3. LOGO ET L'INFORMATIQUE

3.1 *LOGO est un langage structuré*

Pour considérer un langage comme structuré, celui-ci doit répondre à un certain nombre de critères:

- possibilité de créer des modules autonomes mais qui peuvent communiquer entre eux;
- programmation sans GOTO;
- possibilité d'utiliser des variables locales et globales;
- structures de répétition;
- possibilité de structurer les données.

Parmi les langages interprétés, LOGO est l'un de ceux qui répond le mieux à cet ensemble de caractéristiques. Nous l'avons vu, sa structure de procédures et de fonctions qui ont une existence autonome, le rend véritablement modulaire.

De plus, tout argument d'une procédure ou fonction est automatiquement local; toute variable peut être déclarée locale à l'intérieur d'une procédure ou d'une fonction. Enfin, LOGO peut utiliser comme paramètre une liste de listes, ce qui équivaut à pouvoir fabriquer des tableaux.

Du point de vue des structures de répétition, LOGO valorise l'utilisation de la récursivité. Cependant, on peut très facilement créer ses propres structures. Ainsi, un TANT QUE peut se programmer de la façon suivante:

```
POUR TANTQUE :CONDITION :ACTION
  SI NON EXECUTE :CONDITION [STOP]
  EXECUTE :ACTION
  TANTQUE :CONDITION :ACTION
FIN
```

La structure de liste est une façon simple et presque naturelle de représenter les structures de données de type enregistrement. Chaque sous-liste est un autre enregistrement ou un champ.

3.2 *LOGO et la programmation structurée*

La programmation structurée est une approche à la programmation qui est très similaire à une technique de résolution de problèmes. On découpe le problème en sous-problèmes et pour chacun des sous-problèmes, on cherche une structure simple (structure séquentielle, répétitive ou sélective) permettant de résoudre cette partie du problème.

L'utilisation du structurogramme permet de visualiser un projet global comme étant la somme de plus petits projets. Cette structure, qui peut être simulée en Basic, fait partie intégrante du langage LOGO. En effet, ce langage est procédural, c'est-à-dire que l'on peut définir de façon indépendante une série de procédures (mini-programmes) qui, une fois en mémoire, peuvent communiquer les uns avec les autres.

Les procédures peuvent s'appeler mutuellement et se transmettre des données (variables) au niveau local ou global. Les variables locales existent à l'intérieur d'une procédure et lui sont propres; elles peuvent être transmises, mais elles ne sont pas directement accessibles aux autres procédures, contrairement aux variables globales qui sont accessibles à toutes les procédures.

Les variables locales permettent la création d'une nouvelle structure: la structure récursive. Cette structure est très puissante et elle se reconnaît par le fait qu'une procédure se définit en termes d'elle-même. Un bon exemple de structure récursive en mathématiques est la fonction factorielle: le factoriel de 5 est égal à 5 fois le factoriel de 4. Cette structure combinée avec une structure sélective permet des répétitions à plusieurs niveaux pour la valeur des variables locales.

On dit souvent que LOGO est plus qu'un langage, qu'il est presque qu'un didacticiel. Ceci vient du fait que les primitives du langage LOGO sont, pour la plupart, en langage naturel et permettent par le fait même une communication directe avec l'ordinateur. De plus, le fait que plusieurs procédures (programmes) peuvent cohabiter dans la mémoire permet de créer des micro-mondes, c'est-à-dire un milieu contenant le langage LOGO, enrichi de procédures qui permettront de faciliter la réalisation d'un projet. Pour mieux comprendre cette notion de micro-monde, nous aurons plus tard l'occasion d'étudier un micro-monde, celui de la tortue en trois dimensions.

4. GENERALITES SUR LE LOGO

4.1 *Le chargement du langage en mémoire*

Le LOGO[1] n'est pas un langage placé en permanence en MEM. Il doit donc être chargé dans la mémoire (l'espace de travail) de votre ordinateur pour être utilisé. Pour charger le langage LOGO en mémoire, insérez la disquette de LOGO dans l'unité de disque A et allumez l'appareil. Après avoir répondu aux demandes du D.O.S., l'ordinateur chargera le programme GRAPHICS et ensuite le langage LOGO. Vous êtes prêt à commencer à programmer lorsque le texte suivant est affiché à l'écran: COMMENCONS. Vous pouvez maintenant insérer votre disquette dans le lecteur A afin d'y sauver le résultat de votre session de travail.

4.2 *Le vocabulaire*

L'espace de travail est l'ensemble de ce qui est connu de LOGO lorsque vous êtes en interaction avec lui. Le langage LOGO est formé au départ de primitives, qui sont des ordres atomiques (qui ne sont pas composés d'autres ordres). Toutes les primitives font partie du vocabulaire de base de LOGO que vous mettez dans votre espace de travail lorsque vous chargez LOGO. A partir des primitives du LOGO, on peut définir des procédures qui sont une liste de primitives et/ou d'autres procédures qui sont exécutables; on donne à chaque procédure un nom qui lui est propre et qui vient enrichir l'espace de travail. On peut aussi augmenter le contenu de l'espace de travail en ramenant un fichier contenant des procédures créées lors d'une autre session de travail.

Les primitives et les procédures se classent en deux catégories: les commandes, dont l'exécution entraîne une action concrète, et les fonctions, dont l'exécution génère un résultat (objet). De plus, certaines des

---

1. Dans le présente texte, le terme LOGO est utilisé pour décrire le LOGO-TURGEON.

primitives et des procédures peuvent nécessiter des entrées ou paramètres (ex. pour additionner il faut au moins deux nombres). Ces entrées seront des objets appartenant à l'une des trois catégories suivantes:

a) nombre: tout nombre réel;

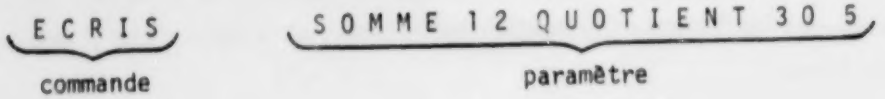
b) mot : toute suite ininterrompue de caractères, excepté les blancs et les caractères identifiant des fonctions +, -, *, /, =, > ou <; tous les mots en LOGO commencent par un " ce qui permet de les distinguer des primitives et des procédures;

c) liste : suite d'objets compris entre crochets ([ ]); les objets peuvent être des mots, des listes ou des nombres. A l'intérieur d'une liste il n'est pas nécessaire de commencer les mots par ".

4.3 *La syntaxe*

La syntaxe des instructions en LOGO est des plus simple; on utilise l'espace vide comme délimiteur entre chacun des éléments. Une instruction est donc une commande suivie de ses paramètres. Un paramètre peut être un mot, une liste ou le résultat d'une fonction qui, elle-même, peut nécessiter certains paramètres. De plus, on définira une expression comme une suite d'instructions valides.

Analysons, par exemple, l'instruction ECRIS SOMME 12 QUOTIENT 30 5. Cette instruction se décompose ainsi:



commande — paramètre

La commande ECRIS nécessite un paramètre qui sera fourni par la fonction SOMME. La fonction SOMME nécessite par ailleurs deux paramètres:

fonction — 1er paramètre — 2e paramètre

On remarque que le deuxième paramètre est aussi une fonction à deux paramètres:

$$\underbrace{\text{QUOTIENT}}_{\text{fonction}} \quad \underbrace{30}_{1^{er}\text{ paramètre}} \quad \underbrace{5}_{2^{e}\text{ paramètre}}$$

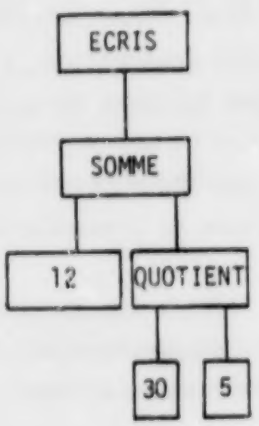
On peut représenter une instruction et ses paramètres par une structure semblable à un structurogramme.

1er niveau: commande ECRIS
1 paramètre

2e niveau: fonction SOMME
2 paramètres

3e niveau: 1 nombre
fonction QUOTIENT
2 paramètres

4e niveau: 2 nombres



EXERCICE

Représentez la structure de l'instruction suivante:

ECRIS PRODUIT SOMME QUOTIENT 80 10 12 a

4.4 *L'éditeur LOGO*

La création de nouvelles procédures, soit à partir des primitives, soit à partir d'autres procédures contenues dans l'espace de travail, est rendue facile par la présence d'un éditeur à l'intérieur duquel on peut travailler d'une façon similaire à un traitement de texte.

On entre en mode éditeur par la commande EDITE. L'ordinateur nous présente alors l'écran éditeur et inscrit le mot POUR dans le coin supérieur gauche. Nous sommes maintenant prêt à donner un nom à la procédure et à inscrire les instructions qui la composeront. Les procédures se présentent de la forme suivante dans l'éditeur: la première ligne contient le nom de la procédure et des variables qu'elle utilise, les lignes suivantes contiennent les instructions et la dernière ligne le mot FIN. On aura donc une structure de présentation comme suit:

```
POUR NOM.DE.LA.PROCEDURE  :VARIABLE1  :VARIABLE2
  Instruction
  Instruction
  ...
FIN
```

On déplacera le curseur dans l'éditeur à l'aide des clés du clavier numérique:

a)

: recule le curseur d'un caractère;

b)

: avance le curseur d'un caractère;

c)

: place le curseur sur la ligne précédente;

d)

: place le curseur sur la ligne suivante.

On corrigera à l'aide des touches:

a) [←] ("backspace"): pour effacer un caractère à gauche du curseur;

b) [DEL] : pour effacer à partir du curseur jusqu'à la fin de la ligne.

Pour sortir de l'éditeur, il existe quatre possibilités:

a) F2 : définit le contenu de l'éditeur et présente un écran divisé;

b) F3 : définit le contenu de l'éditeur et présente l'écran-texte;

c) F4 : définit le contenu de l'éditeur et présente l'écran-graphique;

d) ESC : sort de l'éditeur sans définir son contenu.

4.5 *Les écrans du LOGO*

Il existe quatre types d'écran en LOGO: éditeur, graphique, texte ou mixte. Ces quatre types d'écran se caractérisent de la manière suivante:

a) on entre en écran-éditeur après avoir exécuté la commande EDITE ou la clé F1 (on a déjà parlé de l'éditeur précédemment);

b) le mode texte consacre tout l'écran au texte; on y accède par la commande ECRANTEXTE ou la clé F3;

c) le mode mixte permet simultanément d'écrire du texte et du graphisme sur l'écran (le bas de l'écran est réservé à quelques lignes de texte); on y accède par la commande ECRANDIVISE (ED) ou la clé F2;

d) le mode graphique consacre tout l'écran au graphisme; on y accède par la commande ECRANGRAPHIQUE (EG) ou la clé F4.

5. ACTIVITES DE PROGRAMMATION

5.1 *La géométrie de la tortue*

Le but principal de cette activité est de familiariser le participant avec les commandes de l'éditeur et les notions fondamentales suivantes: procédure, sous-procédure, variable locale, structure séquentielle, répétition conditionnelle et récursive.

Avant de débuter l'activité de programmation, voici une première page d'un dictionnaire LOGO qui contient les commandes de base permettant de réaliser l'activité. Au fur et à mesure que de nouvelles commandes seront utilisées, d'autres définitions seront présentées de cette façon:

Commande	Abréviation	Définition
RECULE n	RE n	recule la tortue de n pas
AVANCE n	AV n	avance la tortue de n pas
DROITE n	DR n	tourne la tortue de n degrés à droite
GAUCHE n	GA n	tourne la tortue de n degrés à gauche
BAISSECRAYON	BC	baisse le crayon de la tortue
LEVECRAYON	LC	lève le crayon de la tortue
GOMMECRAYON	GC	abaisse la gomme à effacer de la tortue
VIDECRAN	VE	vide l'écran graphique et positionne la tortue à [0 0] avec un cap de 0 degré
EDITE	F1	amorce l'éditeur LOGO

Le LOGO étant un langage interprété, il est possible d'essayer les diverses commandes directement du clavier. Ceci vous aidera à vous familiariser avec ces commandes.

5.1.1 *Du triangle au polygone*

A l'aide des commandes du dictionnaire vous devez réaliser les procédures suivantes: TRIANGLE, CARRE, HEXAGONE. Essayez de créer ces formes en mode direct et, ensuite, à partir d'une procédure. Inscrivez le texte de vos procédures dans la grille suivante:

TRIANGLE	CARRE	HEXAGONE

La structure de chacune de vos procédures doit être séquentielle à ce stade-ci, mais on doit voir une régularité qui suggère une structure répétitive. Il existe en LOGO une commande qui permet de réaliser facilement cette structure de répétition.

Commande	Abréviation	Définition
REPETE n liste		exécute n fois les instructions contenues dans la liste

Modifiez vos procédures en utilisant la primitive REPETE et inscrivez le texte de vos procédures dans la grille suivante:

POUR TRIANGLE	POUR CARRE	POUR HEXAGONE
FIN	FIN	FIN

Nous avons enseigné trois nouveaux mots à la tortue ce qui lui permet de dessiner trois figures géométriques différentes. Maintenant, essayons de dessiner la même figure, prenons le triangle comme exemple, mais en trois grandeurs différentes:

```
Petit triangle →REPETE 3[AV  25  DR 120]
Moyen triangle →REPETE 3[AV  50  DR 120]
Grand triangle →REPETE 3[AV 100  DR 120]
```

Nous remarquons que la seule différence (variation) entre les trois instructions est le chiffre qui suit la commande, c'est-à-dire le chiffre qui représente le nombre de pas de la tortue (la longueur du côté du triangle). Ce chiffre étant variable, nous le représenterons par une variable dans la procédure TRIANGLE.

En LOGO, la syntaxe pour l'utilisation d'une variable est très simple. On doit premièrement faire précéder le nom choisi pour la variable d'un deux points (par exemple, :LONGUEUR). Deuxièmement, on doit inscrire dans l'éditeur le nom de cette variable sur la même ligne que le nom de la procédure (par exemple, POUR TRIANGLE :LONGUEUR). Troisièmement, il faut remplacer tous les objets représentés par la variable à l'intérieur de la procédure. Ainsi la procédure suivante:

```
POUR TRIANGLE
  REPETE 3[AV 50 DR 120]
FIN
```

serait écrite de la façon suivante:

```
POUR TRIANGLE :LONGUEUR
  REPETE 3[AV :LONGUEUR DR 120]
FIN
```

Effectuez les mêmes modifications pour les procédures CARRE et HEXAGONE.

Si l'on compare maintenant TRIANGLE, CARRE et HEXAGONE, on retrouve une structure similaire dans les trois procédures:

```
TRIANGLE →REPETE 3[AV :LONGUEUR DR 120]
CARRE    →REPETE 4[AV :LONGUEUR DR  90]
HEXAGONE →REPETE 6[AV :LONGUEUR DR  60]
```

La seule différence entre les procédures est le nombre de fois que l'on REPETE et l'angle qui est donné pour DR (il y a une constance dans le produit de ces deux variables). Créez une procédure POLYGONE générale qui utilisera deux variables: la longueur des côtés et le nombre de côtés. Ecrivez la procédure ici:

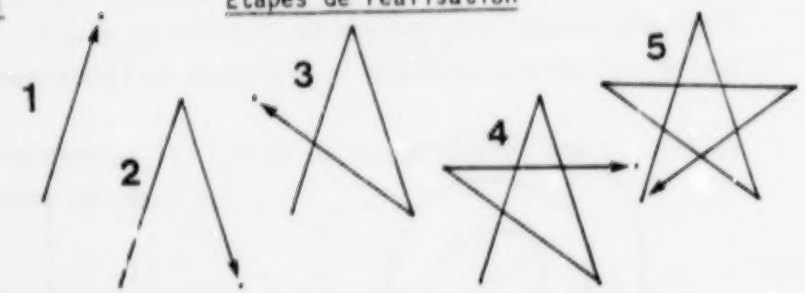
```
POUR POLYGONE


FIN
```

5.1.2 *L'étoile et la spirale*

Pour ce qui est du polygone, notre bagage mathématique nous a permis de déterminer que la somme des angles extérieurs d'un polygone régulier est de 360 degrés. Par contre, la géométrie des étoiles et des spirales ne fait pas partie de notre bagage de connaissances. Nous en connaissons, cependant, la définition globale et les étapes de réalisation. Par exemple, l'étoile à cinq pointes se définit ainsi:

Définition globale

Etapes de réalisation



Nous remarquons que, lors de la réalisation, toutes les étapes successives sont identiques et se présentent comme la réalisation d'un côté. Programmez une procédure réalisant un côté en fonction d'une longueur et d'un angle donné.

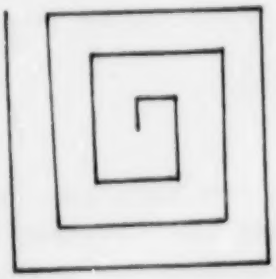
```
POUR COTE :LONGUEUR :ANGLE

FIN
```

Lors de la réalisation de l'étoile, il y a, après l'exécution de chacun des côtés, une décision qui doit être prise pour vérifier si l'on doit faire un côté supplémentaire ou si l'on doit arrêter parce que l'étoile est complète. Les commandes suivantes deviennent nécessaires:

Commande	Abréviation	Définition
SI PREDICAT LISTE1 LISTE2	--	si le prédicat est vrai, la liste 1 est utilisée; sinon, c'est la liste 2
CAP	--	fonction qui retourne la direction de la tortue en degrés

```
POUR ETOILE :LONGUEUR :ANGLE
  COTE :LONGUEUR :ANGLE
  SI CAP=0 [STOP]
  ETOILE :LONGUEUR :ANGLE
FIN
```

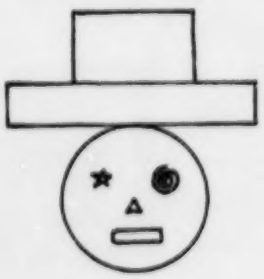
Nous avons utilisé une structure récursive dans la procédure ETOILE, c'est-à-dire que la procédure se rappelle elle-même. Analysez la structure d'une spirale et créez la procédure en conséquence:



```
POUR SPIRALE


FIN
```

5.1.3 *Projet libre*

Vous avez maintenant dans votre espace de travail une quantité de nouveaux mots que vous avez enseigné à la tortue. Utilisez ces nouvelles commandes pout réaliser le projet suivant:



Ramenez le fichier Position .LGO il contient trois procédures utilitaires qui permettent de déplacer la tortue sur l'écran de façon relative:

```
Monte   :N
Glisse  :N
Place   :X :Y
```

Pour ramener un fichier, il suffit de taper RAMENE "Nom.du.fichier. Le fichier se chargera en mémoire.

5.2 La récursivité

La structure récursive est une structure puissante qui permet de créer facilement certains dessins et fonctions mathématiques. Nous avons déjà utilisé la structure récursive pour résoudre le problème de l'étoile et de la spirale. Elle nous a permis d'atteindre un résultat global en répétant une série d'opérations intermédiaires.

Il existe deux familles dans la structure récursive: les récursions avec queue et les récursions sans queue. Les procédures MYSTEREA et MYSTEREB sont deux exemples qui représentent ici ces deux familles:

```
POUR MYSTEREA :N
  SI :N=0 [STOP]
  EC :N
  MYSTEREA :N-1
FIN
```

```
POUR MYSTEREB :N
  SI :N=0 [STOP]
  MYSTEREB :N-1
  EC :N
FIN
```

La procédure MYSTEREA n'a pas de queue: il n'y a pas d'instruction entre le rappel récursif de MYSTEREA et le mot FIN. Par contre, dans MYSTEREB, il y a l'instruction EC :N entre le rappel et le mot FIN. Cette instruction est la queue de la récursivité.

La commande EC objet affiche l'objet à l'écran. Faites vos prédictions en ce qui a trait à l'affichage résultant de MYSTEREA 2 et MYSTEREB 2 avant de lire l'explication suivante.

```
MYSTEREA 2
  TEST FAUX
  AFFICHE 2
  EXECUTE →MYSTEREA 1
           TEST FAUX
           AFFICHE 1
           EXECUTE →MYSTEREA 0
                    TEST VRAI ARRET
           FIN
  RESULTAT 2
           1

MYSTEREB 2
  TEST FAUX
  EXECUTE →MYSTEREB 1
           TEST FAUX
           EXECUTE →MYSTEREB 0
                    TEST VRAI ARRET
           AFFICHE 1
           FIN
  AFFICHE 2
  FIN
 RESULTAT 1
          2
```

5.2.1 *Deux fonctions mathématiques récursives*

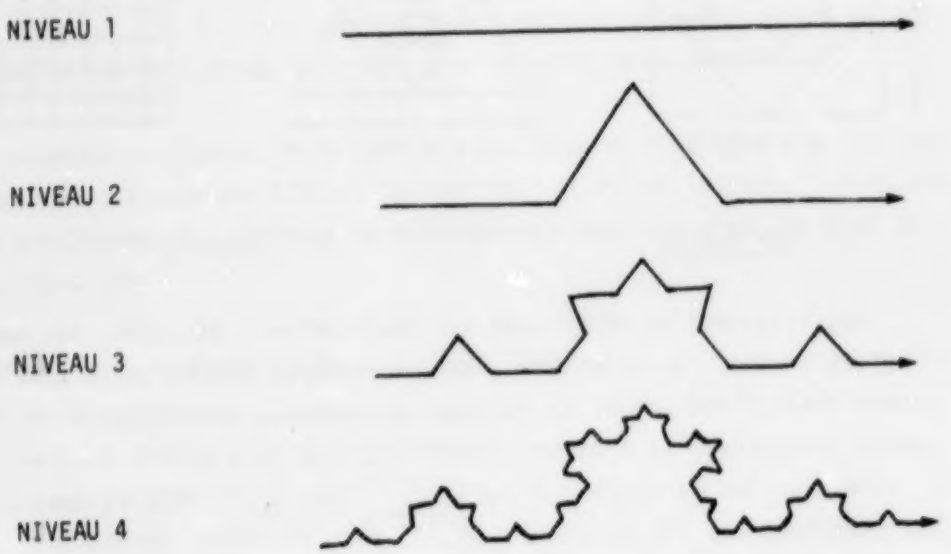
Il existe deux fonctions mathématiques qui se définissent facilement en termes d'elles-mêmes: la fonction factorielle et la fonction exponentielle. Le factoriel d'un nombre peut être défini comme le produit de ce nombre par le factoriel du nombre moins 1 (ex: 5! = 5 X 4!), la décision d'arrêt étant prise par la définition du factoriel qui dit que le factoriel de 1 égal 1. Programmez la fonction mathématique factoriel. Pour ce faire, vous aurez besoin de la fonction LOGO suivante:

Fonction	Abréviation	Description
RETOURNE objet	RT objet	retourne le contrôle à la procédure d'appel et avec objet comme résultat

Programmez la fonction mathématique exponentielle en vous basant sur la définition suivante: $X^n = X*X^{n-1}$ et $X^0 = 1$.

5.2.2 *Les graphiques récursifs*

Pour réussir à programmer facilement un graphique récursif, il faut s'attarder aux niveaux les plus simples du graphique de façon à en saisir la structure. La courbe de KOCH est un exemple de graphique récursif avec queue.

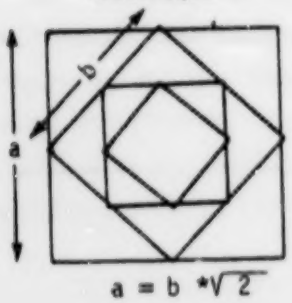


On remarque que le niveau 2 est formé de 4 fois le niveau 1, mais de dimension réduite par un facteur 3. La même chose est vrai pour le niveau 3 qui est formé de quatre fois le niveau 2. En observant les deux premiers niveaux, il est possible de déduire la structure générale de la courbe de Koch et d'en écrire la procédure qui contiendra deux variables: la longueur et le niveau.

```
POUR KOCH :LONGUEUR :NIVEAU

FIN
```

Voici deux autres grahiques qui peuvent être programmés de façon récursive. Le premier peut être programmé avec une récursivité avec ou sans queue. Le second se programme facilement sans récursivité. Cependant, il existe une solution récursive très élégante:

GRAPHIQUE 1

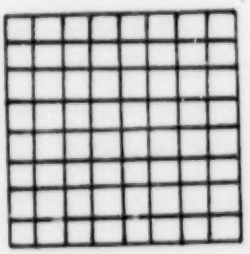


$a = b * \sqrt{2}$

GRAPHIQUE 1a

$a = 2*b$

GRAPHIQUE 2



5.3 *Les listes*

Une liste est un objet LOGO qui peut contenir des mots, des nombres, d'autres listes, des primitives, des procédures, des variables, etc. Cette grande flexibilité permet de realiser de nombreux programmes d'une très grande diversité. La première possibilité que nous allons étudier consiste à faire exécuter le contenu d'une liste. Nous avons déjà vu deux commandes qui exécutent des listes (REPETE et SI) sous certaines conditions. Il existe une commande qui ne fait qu'exécuter des listes:

Commande	Abréviation	Définition
FAIS Liste	--	exécute le contenu de la liste
LISTE objet 1 objet 2	--	crée la liste contenant objet 1 et objet 2
LISCAR	--	retourne le premier caractère lu (sur le clavier par défaut) ATTENDS SI NECESSAIRE
LISLISTE	LL	retourne la ligne tapée au clavier sous forme de liste

5.3.1 *Listes et graphisme*

Ramenez le fichier MICRO.LGO dans l'espace de travail et éditez la procédure MICRO. Regardez sa structure avant de la faire exécuter. En mode d'exécution, les seules touches qui fonctionnent sont les touches A, R, G, D, C, T, O. Utilisez-les pour réaliser un petit dessin et programmez d'autres touches pour enrichir ce petit micro-monde.

Ramenez le fichier AUTO.LGO, qui contient un programme plus puissant, et éditez la procédure AUTO et DEFINITION. Analysez la structure de ces deux procédures et complétez le dictionnaire pour les mots que vous ne connaissez pas.

Commande	Abréviation	Définition

5.3.2 *Listes et texte*

Nous allons donner à l'ordinateur la possibilité de devenir poète en lui fournissant une procédure qui lui permettra de créer des phrases aléatoires. La structure des phrases produites sera très simple. On retrouvera un nom, un verbe et un complément (par exemple, L'homme regarde l'eau).

La première étape est de créer un répertoire de noms, de verbes et de compléments. Pour ce faire, vous utiliserez la commande RELIE pour créer des listes dans chacune des catégories. De même, pour vous faciliter la tâche, vous utiliserez l'éditeur du LOGO.

La deuxième étape est de créer une procédure qui choisit un élément au hasard parmi une liste. On appellera cette procédure: UN :LISTE.

Commande	Abréviation	Définition
HASARD n	--	retourne un entier positif compris entre N et 1 inclusivement
ELEMENT n objet	--	retourne le nième élément contenu dans objet
COMPTE objet	--	retourne le nombre d'éléments contenus dans objet

Etablissez maintenant cette procédure:

```
POUR UN :LISTE

FIN
```

La troisième étape consiste à faire écrire une phrase en utilisant les éléments qui viennent d'être créés.

Commande	Abréviation	Définition
ECRIS objet	EC objet	affiche le contenu de l'objet sans les [ ] de liste
(PHRASE objet 1 objet 2)	(PH objet 1 objet 2)	retourne une liste formée des entrées

Ecrivez cette procédure:

```
POUR POEME

FIN
```

On peut compléter un tel exercice en ajoutant une liste d'adjectifs et en créant une procédure qui accorde les adjectifs avec les noms et les compléments.

6. EXPLORATION D'UN MICRO-MONDE

6.1 *Les micro-mondes et l'apprentissage*

Le monde qui s'offre à nos sens est éminemment complexe. Ce monde est le creuset de tout apprentissage. En agissant sur ce monde et en y réagissant, nous dégageons des principes organisateurs, des lois de transformation, des constantes structurelles et autres composantes d'un modèle de notre environnement que nous nous construisons afin d'apprivoiser sa complexité.

Et si nous apprivoisons sa complexité, c'est pour mieux intervenir sur lui en tant que partie constituante et agissante. En interagissant de façon plus efficace et concertée, nous orientons notre perception et, de ce fait, raffinons notre compréhension de notre environnement. Et le processus se poursuit en se nourissant de lui-même.

L'apprentissage consiste donc à se fabriquer un modèle en perpétuelle évolution de notre environnement. Nos découvertes sont les résultats directs ou indirects de nos agissements sur cet environnement. Une découverte est la prise de conscience d'un principe, d'une loi, d'une constante, etc. La puissance d'une découverte est d'autant plus forte que son champ de validité est vaste. Par exemple, quand un jeune enfant découvre qu'un objet disparaissant de sa vue ne cesse pas pour autant d'exister, il fait alors une découverte majeure puisque cela s'applique à tout objet.

Pour faire une découverte, il faut que notre perception ne soit pas submergée par un ensemble trop vaste de détails. D'où l'utilité de l'attention, qui permet de diminuer le champ de notre perception pour ne laisser passer qu'un nombre limité d'éléments qu'il devient alors possible d'examiner et d'analyser.

Promouvoir l'apprentissage revient à promouvoir la découverte, et cela signifie préparer le terrain à une exploration efficace. Qu'est-ce qui caractérise un bon environnement d'exploration? Sa première qualité est d'être simple: il ne contient qu'un nombre très réduit des composantes du monde réel. Il doit bien sûr être valide; les découvertes qu'y fera l'apprenant doivent s'appliquer au monde réel. Il doit offrir à l'apprenant des moyens d'action; celui-ci doit être appelé à agir sur cet environnement et à constater les résultats de ses agissements. Ces moyens d'action doivent être simples, explicites et peu nombreux. Ils doivent néanmoins permettre un éventail le plus riche possible d'interactions.

Le micro-monde est la réalisation sur médium informatique d'un tel environnement d'exploration. L'ordinateur se prête éminemment bien à cette forme de pédagogie: il est interactif, possède des capacités visuelles et sonores et une grande puissance de traitement de l'information. Il ouvre la porte à des types d'exploration qui étaient auparavant pratiquement impossibles. Le micro-monde que nous allons explorer en est un exemple "vivant".

6.2 *Une description du micro-monde de la tortue tri-dimensionnelle*

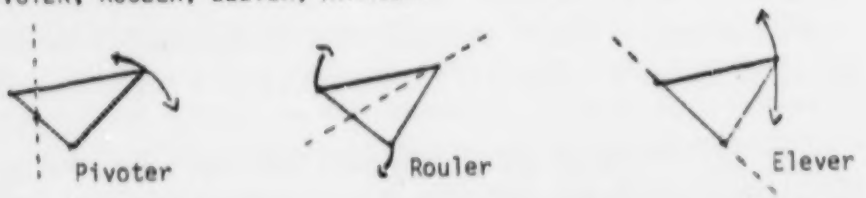
La tortue LOGO a acquis la capacité de se déplacer dans les trois dimensions de l'espace cartésien. Les traits qu'elle dessine dans l'espace tri-dimensionnel sont projetés sur l'écran en respectant les lois de la perspective. De plus il est possible de déplacer le point d'observation dans l'espace tri-dimensionnel. Les commandes qui permettent de déplacer la tortue dans les trois dimensions peuvent être utilisées à l'intérieur de procédures.

6.2.1 *Les objectifs d'apprentissage*

L'exploration formelle de la géométrie tri-dimensionnelle, bien que fondamentale à une bonne appréhension des propriétés de l'espace, ne fait pas partie du curriculum scolaire. La raison principale est le manque

d'outils d'enseignement efficaces. Le micro-monde de la tortue tri-dimensionnelle comble ce vide et permet cette exploration. Voyons les objectifs d'apprentissage que ce micro-monde rend possible:

a) percevoir formellement le mouvement dans l'espace tri-dimensionnel en termes des composantes de base du mouvement, c'est-à-dire PIVOTER, ROULER, ELEVER, AVANCER.



b) percevoir l'image sur l'écran comme une représentation bi-dimensionnelle d'un objet tri-dimensionnel;

c) maîtriser les composantes du mouvement en trois dimensions;

d) créer des trajectoires et définir des objets à partir de ces trajectoires, c'est-à-dire explorer la notion de définition procédurale d'objets tri-dimensionnels;

e) être capable de déplacer le point de vue dans l'espace;

f) comprendre la notion de point de vue; comprendre comment l'apparence d'un même objet varie selon différents points de vue;

g) comprendre que la dimension apparente d'un objet est fonction de l'éloignement du point d'observation;

h) observer les "déformations" dues à la perspective comme fonction de l'éloignement du point d'observation.

6.2.2 *Les pré-requis à l'exploration*

Avant d'aborder ce micro-monde, il faut être familier avec la tortue LOGO et être capable d'écrire des procédures en LOGO. Typiquement, un élève qui travaille en LOGO, qui maîtrise les commandes graphiques de base et qui est capable de définir des figures à l'aide de procédures, est prêt à explorer le micro-monde.

En tenant compte de ce pré-requis, le micro-monde s'adresserait particulièrement bien à des élèves du secondaire. Il peut être soumis à une exploration peu ou très poussée, selon la capacité d'abstraction et de perception de l'élève. Il serait donc possible à un élève de revenir plus d'une fois au micro-monde durant son secondaire. Une compréhension intuitive des angles ainsi que des distances est suffisante pour aborder les composantes du mouvement en 3-D. Après nous être familiarisés avec le micro-monde, il serait intéressant de se demander si ce micro-monde (en partie ou en entier) pourrait s'adresser aussi à des élèves du primaire.

Il faut s'assurer que l'élève soit familier avec un objet défini par ses arêtes, par exemple, un cube fait à partir de tiges. Il serait bon de demander aux élèves de bâtir de tels objets à partir de jeux de construction. L'importance de la manipulation directe ne peut pas être surestimée.

Avant de lancer les élèves dans l'exploration du micro-monde, il est important de simuler en classe avec un objet physique les composantes du mouvement auquel sera soumise la tortue. Avec un objet physique, on simulera les actions de PIVOTER, ROULER, ELEVER et DEPLACER. L'espace de la classe devient l'espace tri-dimensionnel qu'ils sont sur le point d'explorer. La notion de coordonnées cartésiennes en 3-D peut être expliquée en inscrivant sur les murs des axes gradués dont l'origine commune est le centre géométrique de la classe. On demande alors aux élèves de repérer un point dans l'espace défini par trois coordonnées. Puis on leur demande de se déplacer parallèlement à un axe à partir d'un point.

Ensuite, pour développer la notion de point de vue, on accrochera au plafond un objet défini par ses arêtes (une pyramide, par exemple). Puis on donnera aux élèves plusieurs représentations graphiques de l'objet tel que perçu de points de vue différents. On demandera à l'élève de retrouver ces points de vue dans la classe relativement à l'objet suspendu.

Toujours en classe, on abordera le problème de la "disparition" de parties d'un objet causée par un point de vue particulier relatif à l'objet. Par exemple, montrer qu'une tige peut paraître un point si elle est dans le prolongement de la ligne de vue, ou encore démontrer qu'un côté de boîte peut "disparaître" s'il est dans le prolongement de la ligne de vue.

La quantité d'activités hors du contexte informatique peut sembler surprenante. Néanmoins, il ne faut pas oublier que c'est l'apprentissage du monde réel qui est l'objet de cet exercice. L'emploi de l'informatique signifie un transfert d'expériences du monde réel à une représentation du monde réel. Lorsque ce transfert s'est effectué, les découvertes faites à l'intérieur du micro-monde auront une pertinence hors du micro-monde. De plus, celui-ci permettra à l'élève de poser des hypothèses quand au monde réel et de les vérifier formellement à l'intérieur d'un environnement dont il a le contrôle et dont la validité est assurée. Si la connection entre le monde réel et le micro-monde n'est pas établie, son emploi n'a aucune justification.

6.3 *Une aide à l'exploration: les fiches de travail*

L'élève est introduit au micro-monde au moyen de fiches de travail qui joueront le rôle de guide et d'agent provocateur. Les fiches sont divisées en trois recueils:

a) une première approche: une introduction aux composantes du micro-monde;
b) des petites expériences: une série de questions dont le but est d'éveiller la curiosité et d'amener à poser des hypothèses qu'on cherchera à prouver;
c) des idées de projets: des suggestions non-directives de projets d'exploration.

Il serait recommandé de ne remettre qu'un recueil à la fois à l'élève, en lui laissant pourtant le choix du moment (autant que possible) où il passera d'un à l'autre.

Les fiches de travail qui suivent ne sont pas complètes. Elles forment un échantillonnage de ce que pourrait être un support didactique complet du micro-monde de la tortue tri-dimensionnelle. Il serait profitable d'essayer, à la lumière de vos explorations, d'ajouter à ces fiches.

Les fiches du premier recueil ("première approche") devraient être faites ensemble par toute la classe avec l'aide de l'enseignant(e): chacune ne suggère qu'un canevas d'activités que l'enseignant(e) adaptera et complètera selon les besoins et intérêts de sa classe. Quand elles auront été complétées, chaque élève devrait être suffisamment familier avec le micro-monde pour se lancer dans une exploration plus personnelle.

6.3.1 *Une première approche*

Fiche no 1: démarrer le micro-monde de la tortue 3-D

a) Pour mettre en mémoire le micro-monde, fais:

```
? RAMENE "TROISD
```

b) La commande INITIAL permet de commencer à travailler en 3-D; il faut toujours faire INITIAL quand on débute le micro-monde ou quant on veut tout recommencer à zéro. La tortue est devenue une petite boule avec des antennes. Fais:

```
? INITIAL
```

La tortue est à son point de départ; c'est le point (0,0,0). Elle est droit devant toi. Elle est très loin. C'est ainsi à chaque fois que tu fais INITIAL.

c) Fais-la avancer avec la commande AVANCE. Cette commande dit à la tortue d'avancer dans la direction où elle pointe. Fais:

```
? DEPLACE 50
```

Tu as dit à la tortue d'avancer de 50 pas et pourtant on dirait qu'elle n'a pas bougé! C'est parce qu'elle s'est éloignée de toi tout droit. Elle a tracé une ligne. Mais tu ne peux voir qu'un point.

Prends une corde, met-la dans une main et prends un bout dans l'autre main. Pense au bout de la corde comme à la tortue. Fais avancer le bout de la corde. Pense à la corde qui sort de ta main comme au trait que dessine la tortue. Fais sortir un bout de corde de ta main. Ferme un oeil et place la corde devant l'autre. Fais-la tourner jusqu'à ce que tu ne vois plus (que tu caches) le bout et que tu vois la corde le moins possible. La tortue s'est éloignée de toi de la même façon.

d) Tu vas faire pivoter la tortue vers la gauche avec la commande PIVOTE, puis tu la feras avancer. Fais:

```
? PIVOTE 90 DEPLACE 50
```

Tu as fait pivoter la tortue complètement vers sa droite, et tu l'as fait avancer de 50. Prends ta corde et imite ce que viens de faire la tortue.

Fiche no 2: pivoter un peu et pivoter beaucoup

a) Nous allons voir ce qui arrive quand on pivote un tout petit peu la tortue. Remet le micro-monde dans son état de départ en faisant:

```
? INITIAL
```

b) Fais pivoter la tortue un petit peu (par exemple de 10 degrés). Fais:

```
? PIVOTE 10
```

c) Puis fais-la dessiner en la faisant avancer. Fais:

```
? DEPLACE 60
```

Tu ne vois qu'un petit trait. Prends ta corde et imite ce que vient de faire la tortue: fais sortir la corde de ta main en la gardant toujours à la même hauteur mais fais-la sortir un petit peu de travers sur la droite.

d) Remets la tortue à son point initial puis fais-la pivoter plus (prends un chiffre plus petit que 90). Ensuite fais-la avancer. Exécute ces trois commandes:

```
? INITIAL
? PIVOTE 70
? DEPLACE 60
```

Le trait est plus long. Imite la tortue avec ta corde et trouve pourquoi.

Fiche no 3: pivoter de 180 degrés et inverser la direction

a) Repars le micro-monde en faisant:

? INITIAL

b) Fais pivoter la tortue de 180 degrés et fais-la avancer:

? PIVOTE 180 DEPLACE 55

Qu'est-ce qui s'est passé? Peux-tu le montrer avec ta corde? Pivoter de 180 degrés fait faire un demi-tour à la tortue.

Fiche no 4: pivoter d'un angle négatif

a) Repars le micro-monde:

? INITIAL

b) Fais pivoter la tortue de -90 degrés et fais-la avancer:

? PIVOTE -90 DEPLACE 63

La tortue a pivoté de quel côté?

c) Répète l'expérience avec d'autres nombres négatifs.

d) Qu'est qui arrive quand la tortue pivote de -180?

Fiche no 5: reculer

a) Repars le micro-monde:

? INITIAL

b) Fais pivoter la tortue vers la droite un peu:

? PIVOTE 45

c) Fais-la avancer de -70 pas:

? DEPLACE -70

d) Qu'est-ce qui s'est passé? La tortue a bien pivoté vers la droite mais elle a reculé au lieu d'avancer!

e) Trouve comment imiter avec ta corde la tortue qui recule.

Fiche no 6: élever

a) Place la tortue à son point de départ:

```
? INITIAL
```

b) Elève le cap de la tortue avec la commande ELEVE:

```
? ELEVE 56
```

c) Fais-la avancer:

```
? DEPLACE 85
```

Tu as fait monter la tortue.

d) Imite cela avec ta corde.

e) Trouve ce qui se passe quand on élève le cap de la tortue de 180 degrés.

f) Trouve ce qui se passe quand on élève le cap de la tortue d'un nombre négatif.

Fiche no 7: tracer une figure géométrique en 3-D

a) Tu te souviens comment tracer un carré avec la tortue ordinaire (en deux dimensions):

```
REPETE 4 [AVANCE 50 DROITE 90]
```

b) Tu vas faire la même chose sauf que tu vas remplacer DROITE par PIVOTE et AVANCE par DEPLACE. Prends une efface et imite ce que ferait la tortue.

c) Demande à la tortue de faire un carré:

```
? INITIAL
? REPETE 4 [DEPLACE 50 PIVOTE 90]
```

d) Qu'est-ce que tu vois? Et pourquoi?

e) Que peux-tu faire pour mieux voir le carré? Fais-le.

Fiche no 8: voyager autour de la tortue (les commandes)

a) Jusqu'à maintenant, tu as dit à la tortue de bouger tandis que toi tu restais immobile. Imagine que tu es dans un vaisseau spatial et que tu peux le faire voyager autour de la tortue. La commande VOYAGE te permet de faire déplacer ton vaisseau. Fais:

```
? INITIAL
? VOYAGE
```

L'écran a changé. Tu vois maintenant l'écran de commandes. Il te dit beaucoup de choses. Il te donne la position de la tortue:

```
TORTUE:  0, 0, 0
```

Ici la tortue est à (0,0,0). Il te donne aussi ta position:

```
POSITION:  0, 0, -350
```

Tu es donc 350 pas directement derrière la tortue quand tu fais INITIAL

b) Ton vaisseau bouge comme la tortue. Tu peux lever le nez de ton vaisseau en faisant H (pour haut) et le baisser en faisant B (pour bas). Fais H:

```
→ H
```

Tu remarques que la ligne:

```
Elevation 0,  Pivotement 0
```

est devenue:

```
Elevation 10,  Pivotement 0
```

Le nez de ton vaisseau s'est élevé de 10 degrés. Fais baisser le nez de ton vaisseau et observe ce qui se passe:

```
→ B
```

c) Tu peux aussi faire pivoter ton vaisseau à droite (en faisant D) ou à gauche (en faisant G). Observe ce qui arrive quand tu fais:

→ D

→ G

Est-ce que la ligne:

Elévation Pivotement

a changé? Comment?

d) Pour faire avancer ton vaisseau, il faut faire A et pour le faire reculer, il faut faire R. Comme la tortue, ton vaisseau avance dans la direction où il pointe. Fais avancer ton vaisseau puis fais-le reculer et observe comment ta position change:

→ A

→ R

Fais T (terminer) pour retrouver la tortue sur ton écran.

Fiche no 9: voyager (contrôler le déplacement du vaisseau)

a) Tu vas décider quel trajet tu veux faire avec ton vaisseau, par exemple, "monter vers la droite". Décide quelles commandes tu vas utiliser et imite ce que feras ton vaisseau.

Remets ton vaisseau à son point de départ en faisant INITIAL (INITIAL place ton vaisseau ET la tortue chacun à leurs points de départ). Passe aux commandes en faisant VOYAGE:

? INITIAL VOYAGE

Donne les commandes pour faire monter ton vaisseau vers la droite.

b) Fais T pour retourner à la tortue. Sur ton écran tu as déjà remarqué le message:

Je tourne la caméra vers la tortue...

C'est que ton vaisseau est équipé d'une caméra (ton vaisseau n'a pas de hublot, seulement un écran de télévision). Ce que tu vois sur ton écran, quand la tortue apparaît, c'est ce que voit ta caméra.

c) Demande à ta tortue de dessiner un carré (sans changer son élévation):

```
? REPETE 4 [DEPLACE 50 PIVOTE 90]
```

Si tu as défini la procédure CARRE avec un paramètre, tu peux faire:

```
? CARRE 50
```

L'autre fois on ne pouvait pas voir un carré, juste une ligne. Maintenant que tu as changé le point de vue, tu vois le carré au lieu d'une ligne. Pourquoi?

Fiche no 10: faire ROULER la tortue

a) Faire rouler la tortue c'est la faire pencher sur le côté.

b) Avec un objet, examine ce qui se passe quand, après avoir fait ROULER la tortue, on lui demande de s'ELEVER ou de PIVOTER.

c) Repars le micro-monde:

```
? INITIAL
```

et fais rouler la tortue vers la droite:

```
? ROULE 30
```

d) Ensuite trace un carré:

```
? CARRE 67
```

Qu'est-ce qui s'est passé? Pourquoi?

e) Expérimente avec ROULE. Que signifie rouler d'un nombre négatif? Qu'est-ce qui arrive quand la tortue roule de 90, de 180 degrés, de -180 degrés?

Fiche no 11: explorer les commandes MAISON, REDRESSE et NETTOIE

a) Tu peux dire à la tortue de retourner au point (0,0,0) en faisant MAISON. Cette commande ne change pas le cap de la tortue.

b) Fais dessiner ta tortue en la déplaçant dans l'espace, puis fais MAISON:

```
? ELEVE 30 DEPLACE 50 PIVOTE 45 AVANCE 60
? MAISON
```

c) Pour vérifier que la tortue n'a pas changé son cap, fais-la avancer:

```
? DEPLACE 40
```

Est-ce qu'elle a changé de cap?

d) REDRESSE dit à la tortue de prendre le cap qu'elle avait quand tu as démarré le micro-monde (quand tu as fait INITIAL). La tortue ne change pas de position.

e) Fais déplacer ta tortue dans l'espace puis fais REDRESSE:

```
? PIVOTE -47 ROULE 30 DEPLACE 75
? REDRESSE
```

f) Vérifie que la tortue a bien pris le bon cap; fais-la avancer:

```
? DEPLACE 38
```

Comment peux-tu dire qu'elle a repris le même cap que lorsque tu démarres le micro-monde?

g) La commande NETTOIE fais plusieurs choses en même temps. Elle ramène la tortue à (0,0,0), lui redonne son cap du tout début et efface l'écran. C'est comme si tu faisais l'une après l'autre les trois commandes.

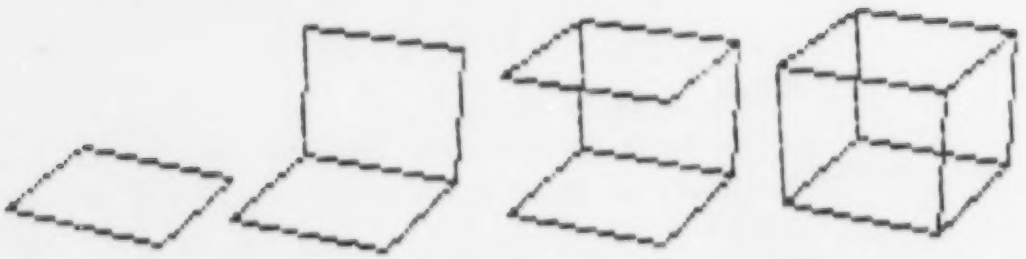
```
? MAISON
? REDRESSE
? VIDECRAN
```

h) Essaie NETTOIE

6.3.2 *Des petites expériences*

Fiche no 12: dessiner un cube

a) On peut construire un cube avec 4 carrés:



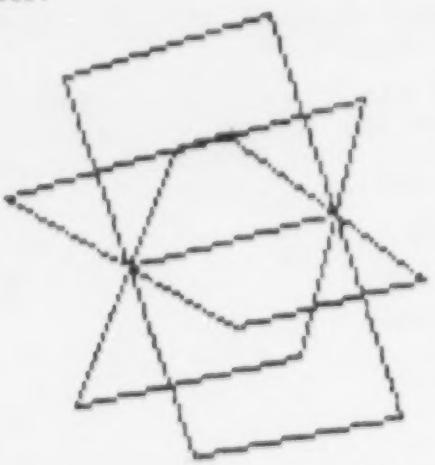
b) De quoi a l'air un cube vu d'en haut? Prends les commandes de ton vaisseau et place toi au-dessus de la tortue. Dessine le cube.

c) De quoi a l'air un cube vu d'en-dessous? Place ton vaisseau au-dessous de la tortue et dessine le cube.

d) De quoi a l'air le cube vu d'en arrière?

Fiche no 14: dessiner une roue à aubes

a) En modifiant la définition du cube, il est possible de dessiner une roue à aubes:



b) Essaie de reproduire ce dessin; pour cela il te faut la bonne procédure ET le bon point de vue.

6.3.3 *Des idées de projet*

Fiche no 15: dessiner une pyramide

a) Tu peux faire un cube, mais sais-tu comment faire une PYRAMIDE? N'oublie pas que la base d'une pyramide est un carré. Réfléchis bien sur la façon dont tu vas construire ta pyramide.

Fiche no 16: dessiner un ressort et une spirale

a) Ce ressort a été dessiné à l'aide des procédures SPIRALE et SPI:

```
TO SPIRALE :COTE
PIVOTE 45
ELEVE 25
ROULE -45
SPI :COTE
END

TO SPI :COTE
DEPLACE :COTE
ELEVE 30
PIVOTE 10
SPI :COTE
END
```

Examine bien ce que font ces procédures.

b) Peux-tu les modifier pour que le ressort soit DROIT et non penché.

c) Cette autre dessin est aussi une spirale. Elle est composée de carrés.

Peux-tu écrire une procédure qui fasse une spirale de carrés? Invente d'autres spirales! Trouve un point de vue qui donne un bel effet.

6.4 *Le résumé des commandes du micro-monde de la tortue tri-dimensionnelle*

a) DEPLACE nombre: fait avancer la tortue du nombre de pas donné;

b) ELEVE nombre : élève le cap de la tortue du nombre de degrés donnés;

c) INITIAL : démarre le micro-monde avec ses valeurs initiales; la tortue est à (0,0,0), le vaisseau est à (0,0,-350), les caps sont identiques et pointent vers le fond de l'écran;

d) MAISON : ramène la tortue au point (0,0,0);

e) NETTOIE : ramène la tortue au point (0,0,0), lui redonne son cap initial et efface l'écran;

f) PIVOTE nombre: fait pivoter la tortue du nombre de degrés donnés:

g) REDRESSE : redonne à la tortue son cap initial;

h) ROULE nombre : fait rouler la tortue du nombre de degrés donnés;

i) VOYAGE : met en mode de déplacement et permet de modifier le point de vue:

- H: élever le cap du vaisseau de 10 degrés,
- B: baisser le cap de 10 degrés,
- D: pivoter vers la droite de 10 degrés,
- G: pivoter vers la gauche de 10 degrés,
- T: orienter la caméra vers la tortue et retourner en mode graphique;

j) primitives LOGO utiles :

- LEVECRAYON LC
- BAISSECRAYON BC
- GOMMECRAYON GC

7. EXERCICE SUPPLEMENTAIRE DE RESOLUTION DE PROBLEMES

7.1 L'acquisition d'habiletés de résolution de problèmes

Le langage de programmation est un outil permettant d'utiliser les ressources de l'ordinateur dans une activité de résolution de problèmes. On n'apprend pas LOGO pour apprendre LOGO et cela est vrai pour tout langage. Le langage est un moyen d'accès privilégié à une utilisation particulière de l'ordinateur: l'ordinateur non pas comme prothèse intellectuelle mais comme instrument de travail et environnement d'exploration.

Un problème se pose très souvent sous la forme de processus (mettre en ordre une liste de noms, transformer des données numériques en graphiques, etc.) dont les solutions sont des procédures. Une procédure est la série de transformations qui mène des données de départ aux résultats finaux. L'ordinateur est une machine qui exécute rapidement et sans erreur des procédures exprimées dans un langage de programmation.

L'activité de résolution de problèmes est le cheminement entre l'exposition du problème et la mise au point de la procédure qui le résoud. Cette activité présuppose des habiletés de résolution de problèmes.

Ces habiletés doivent faire l'objet d'un apprentissage. Quel type d'apprentissage? Puisque la résolution de problèmes est une activité, on acquiert des habiletés de résolution de problèmes en situation d'exploration active. On n'apprend à nager qu'en nageant, pas en en parlant.

Ce n'est pas un apprentissage de mémorisation; être capable de nommer une heuristique (ou stratégie) ne démontre en aucun cas son apprentissage. Ce n'est pas non plus un apprentissage de recettes; mémoriser le mode d'emploi d'une heuristique n'assure pas sa compréhension. De plus, la résolution de problèmes consiste en l'emploi d'un ensemble d'heuristiques dont la composition diffère selon chaque problème à résoudre.

L'apprentissage d'habiletés de résolution de problèmes se vérifie de deux façons: l'intégration d'heuristiques dans un répertoire croissant (intégration suppose compréhension) et une capacité accrue de choisir des stratégies adéquates face à un problème nouveau. On ne peut séparer ces deux ordres d'apprentissage.

Un enseignement visant l'acquisition d'habiletés de résolution de problèmes est voué à l'échec s'il ne tient pas compte de la nature globale de ce type d'apprentissage.

Cet apprentissage doit donc se faire en situation d'exploration active. L'ordinateur est un médium d'exploration privilégié parce qu'il permet à l'apprenant de développer interactivement sa solution à un problème qu'il s'est posé. Par un langage de programmation, l'apprenant enseigne à l'ordinateur comment résoudre un problème précis. Selon les performances de l'ordinateur, l'apprenant est à même de constater les failles de son enseignement et de tenter d'y remédier.

Dans ce tête-à-tête où l'apprenant est le maître (le maître de son apprentissage s'entend), il peut être guidé par un enseignant et mis sur la piste de stratégies dont il a besoin pour réaliser son projet. Il apprend donc des habiletés de résolution de problèmes de la même façon qu'un enfant apprend à parler, non pas pour apprendre à parler mais pour exprimer ce qu'il est.

7.2 *Le problème à résoudre: modéliser le comportement olfactif d'une mouche*

Le problème que vous avez à résoudre dans cet exercice est celui de modéliser le comportement olfactif d'une mouche. Pour résoudre ce problème, il faut réaliser une certaine démarche. A l'intérieur de cette démarche, vous aurez à répondre à des questions et à résoudre des problèmes. Des espaces ont été prévus à cette fin. Ces espaces sont numérotés et, à la fin du texte, des solutions vous sont proposées. De même,

des encadrés contiennent des outils de programmation qui vous seront utiles dans la programmation de vos solutions.

- Enoncer le problème:

  - Il s'agit de modéliser le comportement olfactif d'une mouche.

- Préciser l'énoncé:

  - Qu'entend-on par "comportement"?

    1: _____

  - Qu'entend-on par "olfaction"?

    2: _____

  - Qu'entend-on par "comportement olfactif"?

    3: _____

    _____

  - Qu'est-ce qu'un comportement non-stimulé?

    4: _____

    L'analyse est incomplète mais suffisante pour amorcer la réalisation du projet.

- Trouver un problème moins vaste:

  - Considérons d'abord uniquement le déplacement aléatoire de la mouche

    Avant de s'attaquer à l'ensemble du problème, on tentera de résoudre une version simplifiée. Ensuite seulement on augmentera la solution pour répondre au problème initial.

- Trouver une représentation analogue pour la mouche:

  - La tortue LOGO!

- Trouver les éléments constituants d'un déplacement aléatoire:

  5: _____

  _____

• Simplifier si possible la liste des constituants:

6: _____

• Définir les fonctions aléatoires:

7: _____

_____

• Dégager les variables et les invariants des domaines de variations aléatoires (pour tourner et avancer):

8: _____

_____

PASMAX: _____

ANGLE : _____

• Représenter le problème simplifié par un programme:

- Expliciter les éléments d'un déplacement aléatoire à partir de commandes LOGO:

9: _____

- Trouver une hypothèse de récurrence et déterminer la structure de la procédure LOGO qui fera se déplacer la tortue:

10: _____

_____

_____

_____

- Représenter les fonctions aléatoires en LOGO:

:MOT	: donne la valeur qui est dans la variable MOT (par exemple, ECRIS :PASMAX);
HASARD NOMBRE:	retourne un nombre entier entre 0 (exclusivement et NOMBRE (inclusivement).

11: _____

_____

- Programmer la procédure DEPLACEMENT:

EDITE MOT: permet d'entrer dans l'éditeur avec la procédure MOT (par exemple, EDITE DEPLACEMENT); COMMANDES DE L'EDITEUR: - ESC : sort de l'éditeur sans conserver les modifications; - F3 : sort de l'éditeur en conservant les modifications; - CTRL-K : prélève la portion de texte du curseur à la fin de la ligne, la conserve dans la mémoire-tampon; - CTRL-Y : dépose le contenu de la mémoire-tampon à l'endroit indiqué par le curseur; - ← : efface en reculant.

12: POUR DEPLACEMENT

FIN

• Tester la procédure DEPLACEMENT:

- Donner des valeurs à ANGLE et PASMAX;

RELIE "ANGLE 40: donne la valeur 40 à la variable ANGLE (ne pas oublier le ").

- Appeler DEPLACEMENT (corriger les erreurs de syntaxe s'il y en a);

- Corriger les erreurs de logique (s'il y en a);

- Appeler DEPLACEMENT avec différentes valeurs pour ANGLE et PASMAX; observer les comportements obtenus;

CTRL-BREAK: en mode interactif, cette touche-fonction arrête toute exécution.

- Au lieu d'avoir à entrer "à la main" les valeurs de PASMAX et ANGLE, écrire une procédure INITIAL qui les initialisent interactivement et prépare l'écran;

TAPE:OBJET: écrit l'objet à l'écran, laisse le curseur à la fin de la ligne;
LISLISTE: lit du clavier une LISTE entrée sans crochets délimiteurs et terminée par RETURN;
PREMIER:OBJET: retourne le premier élément de OBJET:
RELIE "PASMAX PREMIER LISLISTE: met dans PASMAX un MOT;
F4: l'écran graphique prend tout l'espace vidéo;
FIXEPAL NOMBRE: sélectionne la palette NOMBRE;
FIXEPC NOMBRE: sélectionne la couleur NOMBRE pour le crayon;
FIXEPAL 1 FIXEPC 3: sélectionne le crayon blanc.

- Faire la liste des problèmes non-prévus:

13: _____

_____

_____

Il faut donc trouver un moyen pour restreindre la tortue à l'intérieur de l'écran.

- Reformuler l'énoncé initial:

14: _____

_____

• Trouver un algorithme pour restreindre la tortue à l'intérieur de l'écran:

15: _____

_____

_____

• Dégager les constituants de cette méthode:

16: _____

_____

_____

_____

• Décrire les arguments, le résultat et les effets secondaires de la fonction OBSTACLE qui testera si un pas donné amènerait la tortue à sortir de l'écran:

17: ARGUMENTS: _____

RESULTAT : _____

_____

EFFETS : _____

• Faire l'analyse modulaire descendante de la fonction OBSTACLE:

- Faire l'analyse en pseudo-langage de la fonction OBSTACLE:

18: POUR OBSTACLE _____

_____

_____

_____

_____

_____

_____

_____

FIN _____

• - Quelles "phrases" ne sont pas immédiatement codables?

19: _____

_____

Ce qui n'est pas immédiatement codable devient un sous-problème.

• Analyser la fonction qui retournera la condition "tortue sortie de l'écran":

- Décrire les arguments, résultats et effets secondaires de la fonction DEHORSP:

20: ARGUMENTS: _____

RESULTAT : _____

_____

EFFETS : _____

- Décrire en pseudo-langage la fonction DEHORSP:

21: POUR DEHORSP _____

_____

_____

_____

FIN _____

- Trouver les coordonnées des limites de l'écran:

22: _____

_____

POS	:	retourne la liste des coordonnées de la tortue;
ECRIS POS	:	écrit sur l'écran les coordonnées de la tortue.

• - Programmer la fonction DEHORSP:

XCOR: retourne l'abcisse de la tortue;

YCOR: retourne l'ordonnée de la tortue;

ABS : retourne la valeur absolue d'un nombre (vous devez l'écrire);

SI COND [LISTE D'INST. "ALORS"][LISTE D'INST. "SINON"]
où COND est une expression booléenne telle que (ABS XCOR) (ABS XCOR) > 120: permet une exécution conditionnelle;

RETOURNE OBJET: permet à une fonction définie de retourner OBJET et d'arrêter sa propre exécution;

Une fonction booléenne retourne soit "VRAI soit "FAUX.

23: POUR DEHORSP

FIN

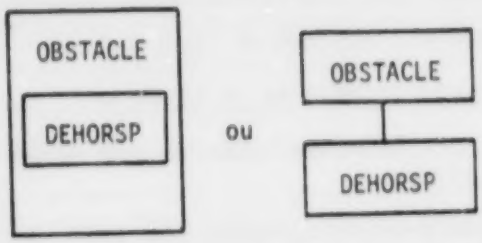
- Tester la fonction DEHORSP; vérifier que DEHORSP retourne la bonne valeur pour chaque type de situation possible.

Une petite procédure telle que:

```
POUR ESSAI :D
  AV :D
  SI DEHORSP alors EC "DEHORS STOP
  ESSAI :D
FIN
```

est utile pour tester une fonction nouvelle.

• Représenter graphiquement la structure de la fonction OBSTACLE:



• Programmer la fonction OBSTACLE:

- Déterminer les variables nécessaires:

24: _____

_____

- Trouver et essayer interactivement la séquence d'ordres en LOGO qui représente la description faite en pseudo-langage de la fonction OBSTACLE:

```
LOCALE "NOM:  déclare la variable NOM locale (par exemple,
              LOCALE "SORTIE);

CT:  cache la tortue;

MT:  montre la tortue;

BC:  descend le crayon de la tortue;

LC:  monte le crayon de la tortue;

RE NOMBRE:  fais reculer la tortue de NOMBRE pas.
```

• - Coder la fonction OBSTACLE:

25: POUR OBSTACLE

FIN

- Tester la fonction OBSTACLE; modifier ESSAI pour tester OBSTACLE;

- Faut-il apporter des modifications à OBSTACLE ou à une autre procédure?

26:

• Intégrer la fonction OBSTACLE à l'intérieur de la procédure DEPLACEMENT en décrivant la nouvelle version de DEPLACEMENT en pseudo-langage:

27: POUR DEPLACEMENT

DEPLACEMENT

FIN

- Se déprendre n'est pas immédiatement codable; c'est un sous-problème.

- Trouver une interprétation pour la procédure DEPRENDRE:

28: _____

_____

_____

- Exprimer la procédure DEPRENDRE en pseudo-langage:

29: POUR DEPRENDRE _____

_____

_____

FIN _____

- Programmer la procédure DEPRENDRE:

30: POUR DEPRENDRE _____

_____

_____

FIN _____

- Tester la procédure DEPRENDRE (en utilisant une version modifiée de ESSAI possiblement).

- Modifier la procédure DEPLACEMENT selon ses nouvelles spécifications:

31: POUR DEPLACEMENT _____

_____

_____

_____

_____

_____

_____

FIN _____

- Tester la procédure DEPLACEMENT:
  - Exécuter la procédure INITIAL, puis exécuter DEPLACEMENT;
  - Répéter en entrant de nouvelles valeurs, observer les comportements qui en résultent.
- Décrire graphiquement la structure de la procédure DEPLACEMENT:

32:

Le problème du déplacement aléatoire est solutionné. Il est maintenant possible de s'attaquer au problème initial en introduisant la dimension olfactive.

Introduire le comportement olfactif comme complexification du problème simplifié du déplacement aléatoire (non-stimulé) de la mouche:

- De quelle manière est-ce que la stimulation olfactive modifie le comportement non-stimulé (remplacement, opposition, perturbation ou autre)?
- Définir les alternatives:

33: REMPLACEMENT: _____

_____

OPPOSITION : _____

_____

PERTURBATION: _____

_____

_____ :

_____

• - Choisir la modification qui constitue la modélisation la plus adéquate; un critère est la simplicité; un autre est la séparation des constituants d'un modèle; il s'agit de opposition.

• Classifier les éléments du problème:

34: STIMULUS : _____

_____

_____

PERCEPTION: modèle différentiel simple; seule la diminution du stimulus est perçue; il y a perception du stimulus quand il y a éloignement. Il n'y a pas de DEGRES de perception; c'est tout ou rien.

On choisit un modèle très simple et on vérifie si le comportement résultant est adéquat; si ce n'est pas le cas, on procède par raffinements successifs.

Donc le comportement stimulé entre en jeu quand il y a augmentation de l'éloignement entre la mouche et la source de stimulus.

• Définir "comportement stimulé":

- Quelle(s) composante(s) du comportement de la mouche se trouve(nt) affectée(s)?

35: _____

_____

- De quelle(s) façon(s)?

36: _____

_____

• Décrire le mécanisme d'opposition entre les comportements stimulé et non-stimulé:

- Un critère décisionnel est de maintenir les deux fonctionnellement distincts. Pourquoi?

37: _____

_____

_____

_____

Le comportement stimulé est suffisamment décrit. Avant de le programmer, il faut d'abord préciser le problème de la perception du stimulus.

- Donner la définition fonctionnelle (applicable dans le modèle informatisé) de la perception:

38: _____

_____

_____

Il faut donc pré-déterminer la position du stimulus sur l'écran pour être capable d'évaluer l'éloignement.

- Ecrire une procédure STIMULUS qui initialisera interactivement la position de la source de stimulus; les variables XSTIM et YSTIM contiendront respectivement l'abcisse et l'ordonnée de la position cartésienne de la source de stimulus:

POINT POSITION	:	fait un point à la position indiquée par la liste POSITION;
POINT LISTE :XSTIM :STIM:		fait un point à (XSTIM, YSTIM).

34: POUR STIMULUS

_____

_____

_____

_____

_____

FIN

- Tester la procédure STIMULUS.

Maintenant que le stimulus a une position déterminée, il est possible d'évaluer la distance entre la tortue et le stimulus pour pouvoir comparer celle avant et celle après un déplacement (ce qui détermine s'il y a perception du stimulus).

- Définir la distance entre deux points (x1,y1) et (x2,y2):

40: distance: _____

- Définir la distance entre deux points (x1,y1) et (x2,y2):

  40: Distance: _____

  Est-il nécessaire d'avoir une évaluation exacte de la distance ou est-ce qu'une évaluation relative est suffisante?

- Trouver une définition plus simple qui satisfait aux exigences du problème:

  41: Distance: _____

- Programmer une fonction DISTANCE qui applique la définition au modèle:

  - Dégager les variables du modèle qui concernent la définition;

XCOR:	retourne l'abcisse de la position de la tortue;
YCOR:	retourne l'ordonnée de la position de la tortue.

  - Coder la fonction DISTANCE (une fonction retourne une valeur):

  42: POUR DISTANCE _____
  _____
  _____
  FIN _____

- Tester la fonction DISTANCE:

  - Exécuter STIMULUS;
  - Déplacer la tortue;
  - Faire ECRIS DISTANCE;
  - Recommencer et vérifier la validité des résultats.

- Décrire en pseudo-langage la condition de perception en terme de la fonction DISTANCE:

  43: Il y a perception quand _____
  _____

- Décrire en pseudo-langage le comportement stimulé de la mouche:

  44: Si _____

  alors _____

- Faire de la dimension variable du comportement stimulé une variable S qui est initialisée dans une nouvelle version de la procédure STIMULUS.

  Maintenant que les constituants du comportement stimulé sont précisés, il est possible de les intégrer dans une nouvelle version de la procédure déplacement.

- Dégager la(les) variable(s) nouvelle(s) nécessaire(s) à l'intégration du comportement stimulé:

  45: _____

  _____

- Décrire en pseudo-langage la nouvelle version de la procédure DEPLACEMENT:

  46: POUR DEPLACEMENT _____

  _____

  _____

  _____

  _____

  _____

  _____

  _____

  _____

  _____

  _____

  FIN _____

- Coder la procédure DEPLACEMENT:

```
47: POUR DEPLACEMENT
    _____________________________________________
    _____________________________________________
    _____________________________________________
    _____________________________________________
    _____________________________________________
    _____________________________________________
    _____________________________________________
    _____________________________________________
    _____________________________________________
    _____________________________________________
    FIN
```

- Tester la procédure DEPLACEMENT:

  - Exécuter la procédure INITIAL;
  - Exécuter la procédure STIMULUS;
  - Exécuter la procédure DEPLACEMENT;
  - Recommencer avec de nouvelles valeurs;
  - Combiner les trois procédures sous une seule nommée mouche pour se simplifier la vie;
  - Observer les relations entre S et les dimensions du comportement aléatoire.

Ceci complète un premier modèle du comportement olfactif d'une mouche. On remarque que des comportements d'apparence complexe peuvent être obtenus à partir de quelques règles très simples. Il semble donc possible à un organisme au système nerveux primitif d'exhiber des comportements complexes. Elaborez sur ce thème.

Le modèle présent se prête très bien à des raffinements successifs. Le premier pourrait être d'ajouter à la perception du stimulus une dimension d'orientation. Postulons que la mouche peut, lors de la perception du stimulus, déterminer si la source de stimulus est à sa gauche ou à sa droite.

Si le "comportement stimulé" consistait à tourner à droite de S degrés, il devient tourner vers la source de stimulus de S degrés.

• Formaliser le comportement stimulé:

- Si le stimulus est perçu à droite, alors tourner de S degrés vers la droite; si le stimulus est perçu à gauche, alors tourner de -S degrés vers la droite.

Ce formalisme indique une solution simple pour amener ce raffinement. Imaginez une fonction ORIENTE qui retourne 1 si le stimulus est perçu à droite et qui retourne -1 s'il est perçu à gauche. Le comportement stimulé devient alors:

TOURNER A DROITE DE ORIENTE * S degrés

Voici le code de la procédure ORIENTE:

```
POUR ORIENTE
  LOCALE "BUT
  LOCALE "DIFF
  RELIE "BUT VERS LISTE :XSTIM :YSTIM
  RELIE "DIFF ABS ( CAP - :BUT )
  TESTE : BUT > CAP
  SI VRAI [ SI :DIFF > 180 [RETOURNE -1] [RETOURNE 1] ]
  SI FAUX [ SI :DIFF > 180 [RETOURNE 1] [RETOURNE -1] ]
FIN
```

VERS POSITION:		donne le cap que devrait prendre la tortue pour pointer vers POSITION;
CAP	:	retourne le cap de la tortue.

• Modifier la procédure DEPLACEMENT pour incorporer la perception de l'orientation du stimulus.

• Tester la nouvelle version de DEPLACEMENT.

Certains animaux primitifs, tels les vers de terre, réagissent aux stimuli en altérant leur vitesse de déplacement et non leur orientation. Quand ils ressentent l'humidité, leur déplacement est inhibé. Ainsi, ils passent plus de temps dans un environnement favorable. Donnons donc à la mouche la capacité supplémentaire de ralentir lorsque l'intensité du stimulus augmente (autrement dit lorsqu'elle s'approche de la source de stimulus).

• Exprimer la notion RALENTIR en termes du modèle:

- RALENTIR affecte quelle composante?

48: _____

- De quelle façon (attention de ne pas perdre la dimension aléatoire)?

49: _____

• Décrire RALENTIR:

- Facteur limitant la longueur maximale d'un pas.

• Trouver les cas limites de RALENTIR:

50: 0 quand _____

1 quand _____

• Trouver une définition fonctionnelle pour RALENTIR:

- RALENTIR = 1 – (:PASMAX / (:PASMAX + DISTANCE))

• Vérifier si la définition de RALENTIR respecte sa description préalable et les cas limites pré-établis:

51: Quand DISTANCE = 0, RALENTIR = _____

Quand DISTANCE = infinie, RALENTIR = _____

• Coder la fonction RALENTIR:

52: POUR RALENTIR _____

_____

FIN _____

• Tester RALENTIR puis modifier et tester la procédure DEPLACEMENT.

Le modèle comporte maintenant un certain degré de raffinement. Vous avez peut-être constaté qu'au fur et à mesure que le modèle se raffinait, il était possible de diminuer la force relative du comportement stimulé tout en préservant un comportement contrôlé par un stimulus. Elaborez.

Finalement, faisons la synthèse du modèle final:

- Le déplacement de la mouche se compose de deux éléments:

53: _____

_____

L'élément TOURNER est sujet à deux influences, aléatoire et stimulée, successivement (donc les influences s'opposent). L'élément AVANCER est aussi sujet à ces deux influences, mais ici l'influence stimulée restreint le champ de variation de l'influence aléatoire: il s'agit donc d'une perturbation.

Plusieurs autres raffinements peuvent être apportés pour rendre le modèle plus réaliste. Une suggestion serait d'introduire la notion d'APPETIT.

• Faire une liste de raffinements possibles:

54: _____

_____

_____

_____

7.3 Les réponses au problème de la modélisation du comportement de la mouche

1: Déplacement

2: Détection d'un stimulus

3: Modification du comportement normal par un stimulus

4: Déplacement aléatoire

5: Tourner, avancer, reculer, sauter...

6: Tourner, avancer

7: Hasard de [-Angle à + Angle] : tourner
Hasard de [Ø à Pas Maximum] : avancer

8: [-Angle... + Angle] — champs de variation
[Invariant Ø... Pas Maximum] — champs de variation
PASMAX : Pas Maximum — variables
ANGLE : Pivotement Maximum (valeur absolue) — variables

9: DROITE, AVANCE

10:
```
REPETER:
  DROITE Hasard [-ANGLE + Angle]
  AVANCE Hasard [ Ø PASMAX]
```

11:
```
:ANGLE - HASARD (2* (:Angle + 1))
HASARD (:PASMAX + 1)
```

12:
```
POUR DEPLACEMENT
  AVANCE HASARD (:PASMAX + 1)
  DROITE (:ANGLE - HASARD (2* :Angle + 1))
  DEPLACEMENT
FIN
```

13: La tortue sort de l'écran!

```
14: Modéliser le comportement olfactif d'une
    mouche à l'intérieur d'une enceinte

15: Envoyer la tortue en éclaireur
    invisible pour tester s'il y a sortie
    Si une sortie est anticipée, alors se déprendre

16: Eclaireur
    Tester sortie
    Se déprendre

17: Arguments: pas proposé
    Résultat: Vrai: si sortie
              Faux: si pas de sortie
    Effets: Aucun

18: Pour OBSTACLE sur un pas
      Rendre tortue invisible
      Rendre tortue non-traçante
      Avancer du pas donné
      Vérifier si sortie
      Le retenir
      Reculer du pas
      Rendre tortue visible et traçante
      Retourner s'il y a eu sortie (True ou False)
    FIN

19: Vérifier si sortie

20: Arguments: Aucun
    Résultat: True si hors limites
              False si dans limites
    Effets: Aucun

21: Pour DEHORSP
      Si position en x HORS limites alors retourne True
      Si position en y HORS limites alors retourne True
      Retourne False
    FIN
```

22: x: -123 à +123
y: -88 à +88

23: POUR DEHORSP
SI (ABS XCOR) > 123 [RETOURNE "VRAI]
SI (ABS YCOR) > 88 [RETOURNE "VRAI]
RETOURNE "FAUX
FIN

24: Sortie

25: POUR OBSTACLE :PAS
LOCALE "SORTIE
CT PU
AV :PAS
RELIE "SORTIE DEHORSP
RE :PAS
MT BC
RETOURNE :SORTIE
FIN

26: FENETRE pour permettre à la tortue de sortir de l'écran

27: POUR DEPLACEMENT
Tourner à droite d'un angle entre -Angle et +Angle
Choisir un pas entre 0 et Pasmax
S'il y a obstacle pour ce pas alors se déprendre
Sinon avancer du pas
DEPLACEMENT
FIN

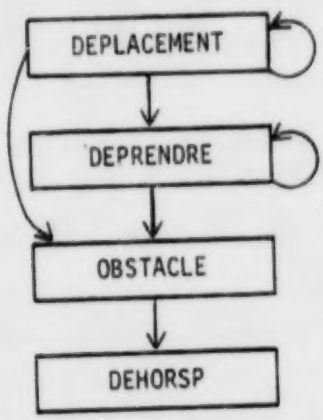
28: Tourner à droite jusqu'à ce
qu'il n'y ait pas d'obstacle
pour le pas souhaité

29:
```
POUR DEPRENDRE :Pas
  A droite de 10 degrés
  S'il y a obstacle pour le pas alors recommencer la procédure
FIN
```

30:
```
POUR DEPRENDRE :PAS
  DR 10
  SI OBSTACLE :PAS [DEPRENDRE :PAS] [AV :PAS]
FIN
```

31:
```
POUR DEPLACEMENT
  LOCALE "PAS
  DR (:ANGLE - HASARD (2* (:ANGLE + 1)))
  RELIE :PAS HASARD (:MAXPAS + 1)
  SI OBSTACLE :PAS [DEPRENDRE :PAS] [AV :PAS]
FIN
```

32:



33:

Remplacement: qui prend la place du comportement aléatoire

Opposition : qui s'ajoute au comportement aléatoire

Perturbation: qui modifie le comportement aléatoire

34: Stimulus: - décrit par sa position
- intensité constante

35: Tourner

36: Tourne dans une direction

37: Aléatoire: Avancer ? [0 PASMAX]
Tourner ? [-ANGLE +ANGLE]
Stimulé : Tourner de S (constante)

38: Eloignement avant déplacement < éloignement après
Déplacement

39: POUR STIMULUS
VE
-message à l'usager-
RELIE "XSTIM PREMIER LISLISTE
- etc...-

FIN

40: Distance = $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

41: Distance <--> ABS $(x_1 - x_2)$ + ABS $(y_1 - y_2)$

42: POUR DISTANCE
RETOURNE (ABS :XSTIM - XCOR) (ABS :YSTIM - YCOR)
FIN

43: Il y a perception quand distance avant < distance après

44: Si distance avant - distance après
alors tourner à droite de S

45: Avant

```
46: POUR DEPLACEMENT
    Calculer et retenir distance dans avant
    Tourner à droite de ? [ANGLE + ANGLE]
    Choisir un pas de ? [Ø MAXPAS]
    Tester s'il y a obstacle
    Si Faux alors avance du pas
    Si Vrai alors se déprendre
    Si distance avant < distance après
      alors tourner à droite de S degrés
    Recommencer
    FIN

47: POUR DEPLACEMENT
    (LOCALE "PAS "AVANT)
    RELIE "AVANT DISTANCE
    DR (:ANGLE - HASARD (2* (:ANGLE + 1)))
    RELIE :PAS HASARD (:MAXPAS + 1)
    SI OBSTACLE :PAS [DEPRENDRE :PAS] [AV :PAS]
    SI :AVANT < DISTANCE [DR :S]
    DEPLACEMENT
    FIN

48: AVANCER

49: Restreindre le champ de variation

50: 0 quand  distance = Ø
    1 quand  distance = infinie (très grande)

51: Quand distance = 0, ralentir = Ø
    Quand distance = infinie, ralentir = 1

52: POUR RALENTIP
      RETOURNE (1 - (:PASMAX / (:PASMAX + DISTANCE)))
    FIN
```

53: Avancer

Tourner

54: Laissez libre cours à votre imagination